IES de Teis
Avda. de Galicia, 101
36216 – Vigo

Tfno.: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

Réplicas en MongoDB

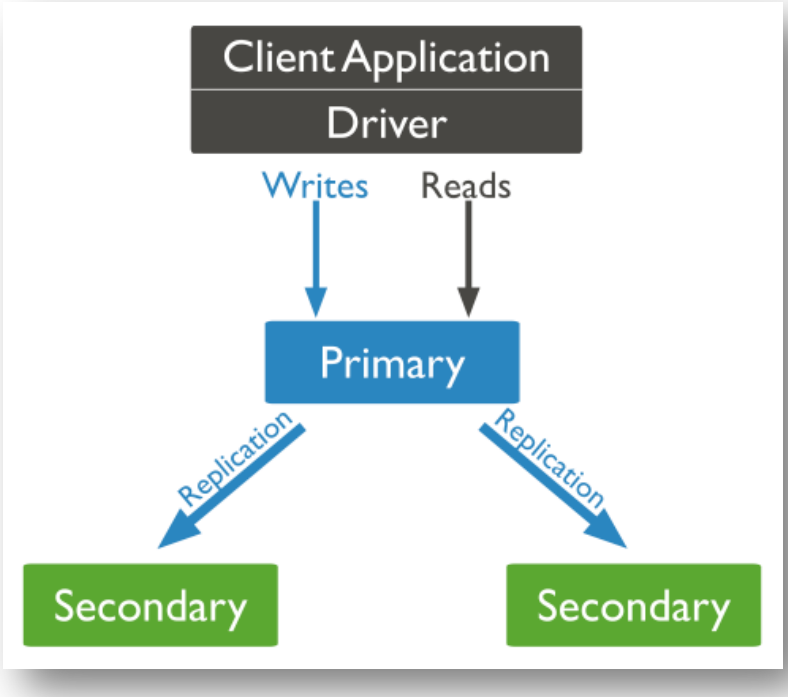
Definiremos un conjunto de réplicas en MongoDB como un grupo de servidores con procesos mongod que almacenan el mismo conjunto de datos.

Los conjuntos de réplicas suponen la herramienta de MongoDB para proporcionar redundancia y, por lo tanto **tolerancia a fallos** ante el fallo de algún servidor. Dependiendo de la configuración, las réplicas también ayudan a la **alta disponibilidad**.

Recuerda que la alta disponibilidad procura que los datos siempre estén accesibles incluso cuando hay una falta de disponibilidad en el sistema

De entre los nodos que componen un conjunto de réplicas uno y solo uno tendrá el papel de primario y el resto se consideran secundarios.

El **nodo primario** se encargará de todas las **operaciones de escritura** en la base de datos. Por un lado, MongoDB aplicará las escrituras en la base de datos del nodo primario y por otro lado dejará la orden de escritura en un registro de operaciones llamado ”oplog“.



Por defecto, el nodo primario también se encarga de todas las operaciones de lectura pero se puede cambiar la configuración para que las lecturas se realicen en los nodos secundarios.

Los nodos secundarios se encargan de mantener una réplica de la base de datos del nodo primario. Para realizar esto reproducen de manera asíncrona las operaciones de escritura que hace el nodo primario, y que este va almacenando en un registro de operaciones llamado "oplog", en sus propios datos.

El "oplog" almacena las operaciones que alteran la base de datos y en momento en que se hacen en una estructura circular donde los registros nuevos sobrescriben los más viejos. Por defecto, el tope de espacio que puede ocupar un "oplog" es de 50 gigabytes. Podemos obtener información administrativa del oplog usando el comando rs.printReplicationInfo(),

Gracias a esto se puede recuperar el sistema desde un momento determinado, comprobar su hay mucho retraso entre el primario y los secundarios o recuperar es estado después de una operación de mantenimiento de un nodo. Para consultar y comparar los oplogs de los nodos secundarios con el nodo primario se puede usar el comando rs.printSecondaryReplicationInfo().

Los "oplogs" de cada nodo se pueden consultar en la base de datos "local" dentro de la colección "oplog.rs". Los comandos guardados en "oplog" son idempotentes. Esto quiere decir que se pueden aplicar una o varias veces que siempre obtendremos el mismo resultado.

Por ejemplo, si actualizamos el valor de un campo de un registro aumentando en uno su valor. Lo que se guardará en el oplog es la actualización al valor X. De esta manera, aunque apliquemos varias veces la operación siempre dará el mismo resultado.

Si en algún momento el nodo primario deja de estar disponible, alguno de los secundarios será elegido para tomar su puesto y convertirse en el nuevo primario en un proceso que se llama "**elección**".

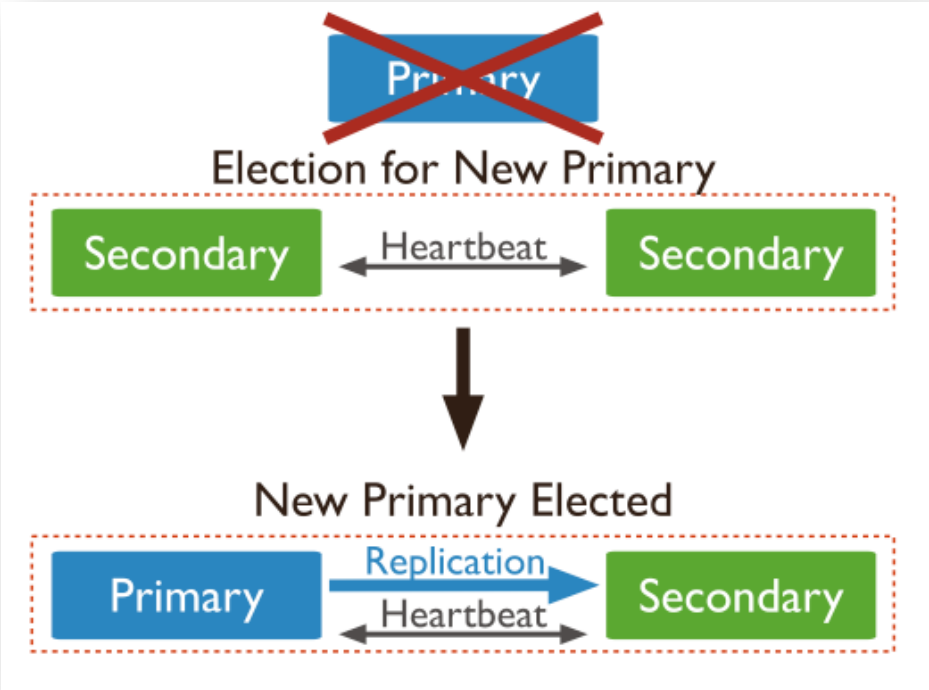
Llamamos "**failover**" al proceso de elección de un nuevo primario cuando el anterior deja de estar disponible y su puesta en marcha. Este proceso se desencadena automáticamente en cuanto el primario deja de estar disponible.

La elección de un nuevo primario no solo se desencadena ante la pérdida de comunicación durante 10 segundos con el primario actual, también se elige en caso de añadir un nuevo nodo al conjunto de réplicas, al iniciar el conjunto de réplica o también cuando se degrada un primario a un secundario con un rs.stepDown() o se hace una reconfiguración del conjunto con un rs.reconfig().

La elección para que un nodo secundario pase a ser el único primario del conjunto de réplica es algo más complicado de lo que parece. Es necesario gestionar los empates cuando hay un número par de secundarios y también es necesario gestionar qué pasa cuando un conjunto grande de nodos queda

partido por la mitad ya que, en este caso puede que cada mitad elija su propio primario cuando solo puede haber uno.

En la elección de un nuevo primario los secundarios se votan teniendo en cuenta la antigüedad de su réplica más actualizada, las latencias de red y otros factores de los que se informa en los latidos. El administrador de red también puede forzar la elección de un primario de entre los secundarios disponibles asignando un nivel de prioridad entre 0 y 1000. Un nivel 0 de prioridad convierte a un nodo secundario en inelegible para ser un nodo primario. Cuanto más alta sea la prioridad asignada a ese nodo, más opciones de convertirse en un nuevo primario tendrá. Por defecto todos los nodos tienen una prioridad de 1.



Es habitual encontrar conjuntos de réplica de 3, 5 o 7 nodos. Preferentemente elegimos un número impar de nodos para que el sistema de elección siempre tenga una mayoría clara.

Si imaginamos un conjunto de réplicas cuyos nodos se encuentren geográficamente distribuidos podríamos tener las siguientes situaciones:

- Dos nodos en el datacenter A y un nodo en el datacenter B.
  - Si cae el datacenter A, el conjunto de réplicas quedará como de solo lectura porque un único nodo no se puede nombrar a si mismo primario.
  - Si cae el datacenter B, el conjunto de réplicas sigue funcionando en lectura y escritura..

- Un nodo en el datacenter A, otro nodo en datacenter B y el tercer nodo en el datacenter C.
    - Si cae cualquiera sigue funcionando, los otros dos nodos pueden elegir un primario si es necesario.
- Tres nodos en el datacenter A y 2 nodos en el datacenter B.
    - Si cae el datacenter A, el conjunto de réplica queda como solo lectura ya que con 2 nodos (de 5) nunca habrá mayoría suficiente para elegir primario.
    - Si cae de datacenter B, los 3 nodos del otro data center son suficientes para elegir un nuevo primario si hiciera falta.
- Dos nodos en el datacenter A, dos nodos en el datacenter B y un nodo en el datacenter C.
    - Si cae cualquier datacenter, los nodos del resto de los datacenters son suficientes para elegir un primario si fuera necesario.

En todos los casos lo importante es la ubicación de nodo primario porque es el único que puede realizar operaciones de escritura.

Los nodos del conjunto de réplicas informan de su estado mediante latidos. Es un concepto que ya conocemos de HDFS que significa el envío periódico de información de estado entre los nodos.

Los latidos que emiten los nodos son, por defecto, cada 2 segundos. Para nosotros será suficiente con saber que ante la ausencia de un nodo primario durante más de 10 segundos por defecto, los nodos secundarios se pondrán de acuerdo para que alguno de ellos ocupe el puesto de nodo primario. En general no deberían pasar más de 12 segundos entre detectar la pérdida del nodo primario y que un secundario retome su función. Mientras no tengamos un nodo primario nombrado no se admitirán operaciones de escritura en la base de datos.

Los conjuntos de réplicas pueden tener hasta 50 nodos secundarios de los cuales, solo 7 pueden votar en la elección del nuevo nodo primario si hiciera falta. Los restantes nodos secundarios serán o no votantes o de prioridad 0. La diferencia reside en que los nodos con prioridad 0 no son candidatos para ser el nuevo nodo primario.

Operaciones de lectura.

Por defecto las lecturas se realizan en el nodo primario del conjunto de réplica. Los clientes pueden especificar sus preferencias de lectura de entre las siguientes:

- Primary. Es el modo por defecto donde las lecturas se realizan en el nodo primario.
- PrimaryPreferred. Las lecturas se realizarán sobre el nodo primario pero si este no estuviera disponible, se podrían hacer lecturas de los nodos secundarios.

- Secondary. Las lecturas se realizarán en cualquiera de los nodos secundarios.
- SecondaryPreferred. Las lecturas se hacen desde nodos secundarios. En el caso de que no quedara ningún nodo secundario entonces se podría leer desde el nodo primario.
- Nearest. La lectura se hace en el nodo más cercano atendiendo a su latencia. No importa si es secundario o primario.

Hay que tener en cuenta que la replicación se lleva a cabo de manera asíncrona por lo que las lecturas de los nodos secundarios puede que no reflejen aun el valor actual real del nodo primario.

Operaciones de escritura.

En cuanto a la escritura podemos configurar cuando MongoDB dará por concluida una operación de escritura en la base de datos. Por defecto se dará por hecha una operación de escritura cuando una mayoría calculada de los nodos del conjunto de réplica actualicen sus datos de "oplog". Esto resulta interesante cuando una red queda dividida en dos y el nodo primario queda en el lado de la minoría. En base a los latidos que recibe pronto se dará cuenta que no cuenta con la mayoría y se promocionará solo para ser un nodo secundario. Por otro lado, en la partición mayoritaria algún nodo habrá tomado el papel de primario. Aunque durante uno instante pueda haber dos primarios en el conjunto de réplica solo uno de ellos podrá devolver las confirmaciones de escritura por mayoría.

También podemos indicar el número de nodos con el que nos damos por satisfechos. Con un "1" estaremos indicando que con escribir en el nodo primario es suficiente. Con "3" estaremos indicando que daremos por acabada la operación de escritura en cuando el nodo primario y dos secundarios actualicen sus datos.

Desplegar un conjunto de réplicas.

El modo preferido de gestionar un clúster de MongoDB es usar nombres de hosts en lugar de direcciones IP. Así, la responsabilidad de las IPs que puedan cambiar es del DNS y permanece transparente a MongoDB.

Por ello el primer paso consiste en configurar los nodos con una IP fija, con un nombré único en el archivo "/etc/hostname" y con el resto de IPs-nombres de los nodos del conjunto de réplica en el archivo "/etc/hosts". También tendremos que instalar un servidor MongoDB, preferiblemente como servicio y que lea su configuración desde su archivo en /etc..

Modificaremos el archivo de configuración para añadir un campo con una clave "replSetName" y un valor de la cadena de texto con la que llamaremos a la réplica.

En una única instancia mongod del conjunto de réplica ejecutaremos el comando "initiate()" al que le pasaremos un documento con dos campos.

En el campo con clave "_id" pondremos como valor el mismo nombre que le dimos a la réplica en el archivo de configuración.

En el campo con clave "members" pondremos como valor un array con tantos documentos como nodos tenga el conjunto de réplica. Cada uno de estos subdocumentos tendrá un campo con clave "_id" secuencial empezando por 0, y un campo con clave "host" cuyo valor será una cadena con el nombre y puerto del nodo.

```
rs.initiate({
        "_id": "rs0",
        "members": [
                { "_id":0 , "host":"nodo1:27017" },
                { "_id":1 , "host":"nodo2:27017" },
                { "_id":2 , "host":"nodo3:27017" }
                ]
        })
```

Con el comando anterior, MongoDB arrancará un conjunto de réplica con los nodos indicados.

```
rs.conf()
```

Con este comando, MongoDB devolverá la información de configuración del conjunto de réplica como los tiempos entre latidos, timeouts o los datos de los nodos referentes a sus votos.

```
rs.status()
```

Este comando devuelve información mucho más detallada sobre el conjunto de réplicas indicando entre otros muchos parámetros que nodo es el primario y cuales los secundarios.

Eliminar un nodo del conjunto de réplicas.

En primer lugar debemos apagar el nodo o, al menos la ejecución de su instancia de mongod. Podemos también conectarnos al shell de ese nodo y ejecutar el siguiente comando:

```
db.shutdownServer()
```

En segundo lugar tendremos que averigua cual es el nodo primario en este momento. Para ello podemos interpretar la salida del siguiente comando ejecutado desde cualquiera de los nodos.

```
db.hello()
```

Por último, nos conectamos al nodo primario y desde allí ejecutamos el comando "remove" al que pasaremos como parámetro una cadena con el nombre o ip y puerto del nodo que queremos dar de baja.

```
rs.remove("nodo2:27017")
```

Añadir un nodo a un conjunto de réplica.

En caso de incorporar un nuevo nodo al conjunto de réplica, este permanecerá unos segundos en estado "STARTUP2" que indica que está clonando las bases de datos alojadas en el nodo primario y actualizando las últimas operaciones de escritura que figuren en "oplog". Al terminar pasará a un estado "SECONDARY" y a partir de ahí seguirá actualizando según las operaciones de "oplog".

En primer lugar hay que arrancar el servicio mongod en el nodo que queremos incluir asegurándonos que en su archivo de configuración se hace referencia al nombre del conjunto de réplica mediante la clave "replSet".

En segundo lugar hay que averiguar que nodo es el primario y para ello interpretamos la salida del comando "db.hello()".

Una vez conectados al nodo primario ejecutamos el comando "add" pasándole como parámetro un documento cuya clave es "host" y su valor el nombre y el puerto del nuevo nodo a añadir.

```
rs.add( { "host":"nodo4:27017" } )
```

Otra opción para añadir o eliminar un nodo es usar el comando "reconfig" al que tendremos que pasarle un documento como el que le pasamos a "rs.initiate()" o también la salida del comando "rs.conf()".

Configuración inicial conjunto de réplicas en MongoDB

Partimos de 3 servidores MongoDB instalados y bien configurados.

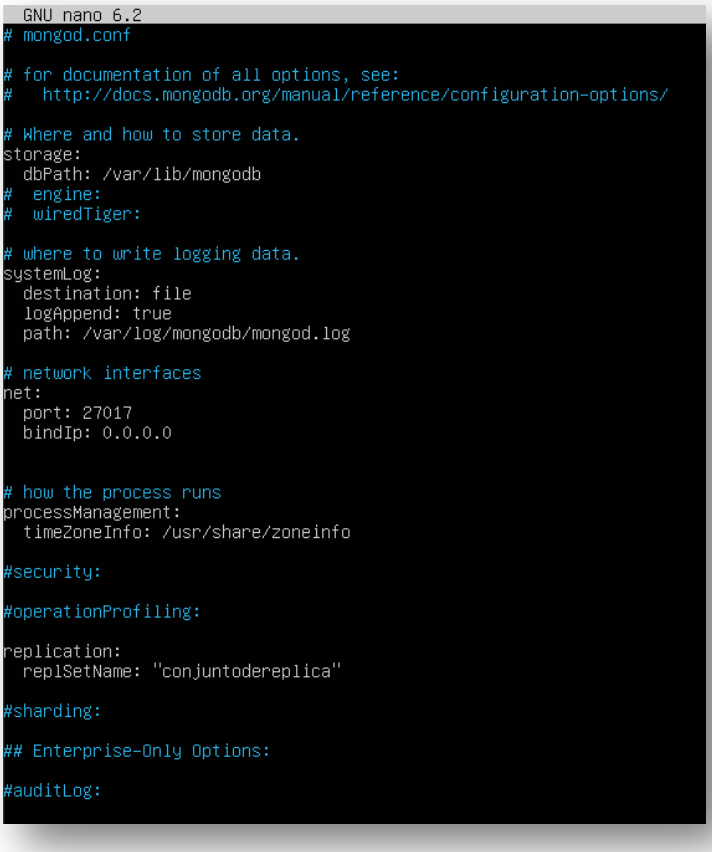
En primer lugar debemos editar el archivo *"/etc/mongod.conf"* con el siguiente comando:

```
sudo nano /etc/mongod.conf
```

En este archivo debemos añadir una sección llamada "replication" y dentro de ella un campo *"replSetName"* con el nombre que identificará a este conjunto de réplica.

IES de Teis
Avda. de Galicia, 101
36216 – Vigo

Tfno.: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es



Es importante revisar cada archivo de configuración. Los errores típicos tienen que ver con el uso incorrecto de mayúsculas y minúsculas. También es habitual encontrar errores de indentación. En el fondo esto es un archivo YAML y los espacios son importantes.

Para que el archivo de configuración se vuelva a leer es necesario reiniciar el servicio de MongoDB en todos los nodos que hemos modificado. Lo podemos hacer con el siguiente comando en cada nodo:

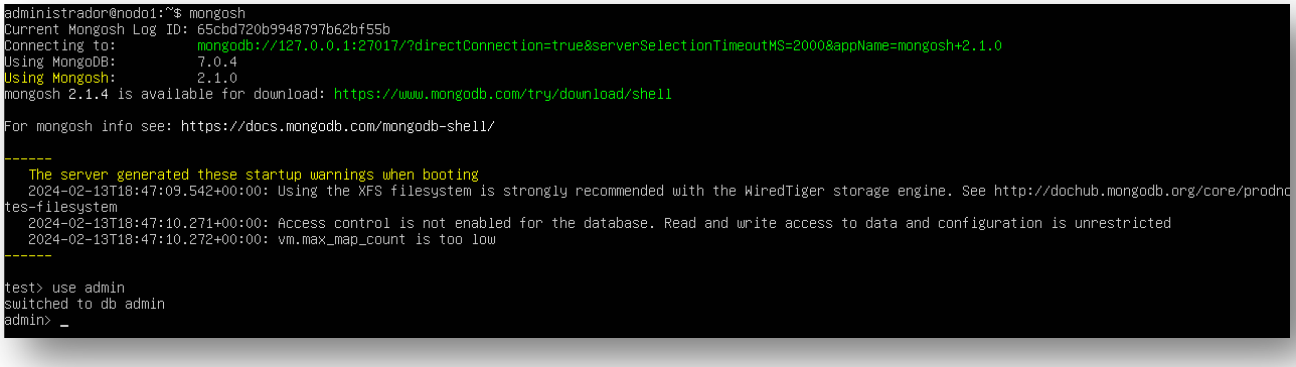
```
sudo systemctl restart mongod.service
```

Nos conectaremos a uno de los nodos del conjunto de réplicas mediante un mongo Shell. La forma más sencilla es desde el propio nodo ejecutar el siguiente comando

```
mongosh
```

Para conectar a los nodos que formarán parte del conjunto de réplicas debemos modificar la base de datos "admin" así que es necesario cambiar la base de datos activa a esta con el comando:

```
use admin
```



Veremos que en el prompt ya se muestra el nombre de la base de datos activa, ”admin”.

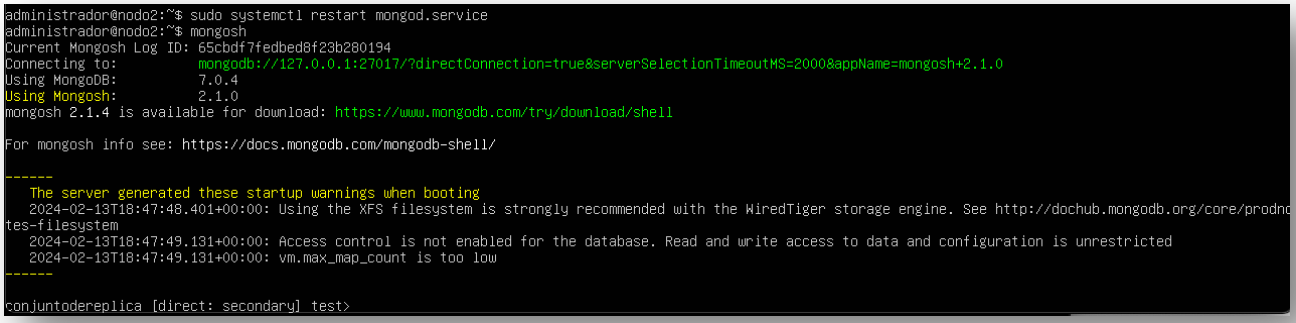
Ahora ejecutaremos el comando ”rs.initiate()” donde pasaremos por parámetro un documento con un ”_id” cuyo valor es el nombre del conjunto de réplica, un campo ”members” con un array de documentos, uno por cada nodo y finalmente un campo versión que permitirá hacer un seguimiento si el conjunto de réplicas va modificando el número de nodos que lo componen.

```
rs.initiate({
        “_id”:”conjuntodereplica”,
        “members”:[
                {“_id”:0, “host”:”nodo1:27017” },
                {“_id”:1, “host”:”nodo2:27017” },
                {“_id”:2, “host”:”nodo3:27017” }
                ],
        “version”: 1
        })
```

Lo que estamos haciendo aquí es configurar el estado inicial del conjunto de réplicas informando de los nodos que lo componen. Este sería un buen momento para asignar prioridades a los nodos o bien definir las restricciones y preferencias en las lecturas y escrituras.

Si nos fijamos en el prompt, en el primer momento después de definir el conjunto de réplicas nos dice que tenemos una conexión directa con "other". Si esperamos un par de segundos y volvemos a pulsar Intro veremos que ahora indica que estamos conectados directamente a un nodo primario. Es el tiempo que ha llevado la elección del nodo primario en el primer arranque.

Si nos conectamos a otro nodo y desde allí abrimos un mongosh también nos aparecerá que estamos conectados directamente a un secundario.



En la imagen podemos ver como desde nodo2 el prompt nos indica que estamos conectados directamente a un secundario.

Para comprobar el estado del conjunto de réplica podemos usar el siguiente comando:

```
rs.status()
```

En esta captura podemos ver la información del nodo3 como que figura como "SECONDARY" y que el nodo origen desde el que sincroniza, el nodo primario, es el nodo1.

Para acceder al "conjunto de réplicas" y no a un nodo específico debemos hacer una conexión a MongoDB usando una cadena de conexión como ya hemos utilizado con Atlas. En esta cadena de conexión deben aparecer todos los nombres y puertos de nodos separados por comas, que componen el conjunto de réplicas y también el nombre con el que hemos definido el conjunto de réplicas.

```
mongosh "mongodb://nodo1:27017,nodo2:27017,nodo3:27017/?replicaSet=conjuntodereplica"
```

Haciendo un pequeño abuso del lenguaje podemos aprovechar que usamos los puertos por defecto y que es el único conjunto de réplica que tenemos para que la cadena de conexión quede mucho más sencilla.

```
mongosh "mongodb://nodo1,nodo2,nodo3"
```

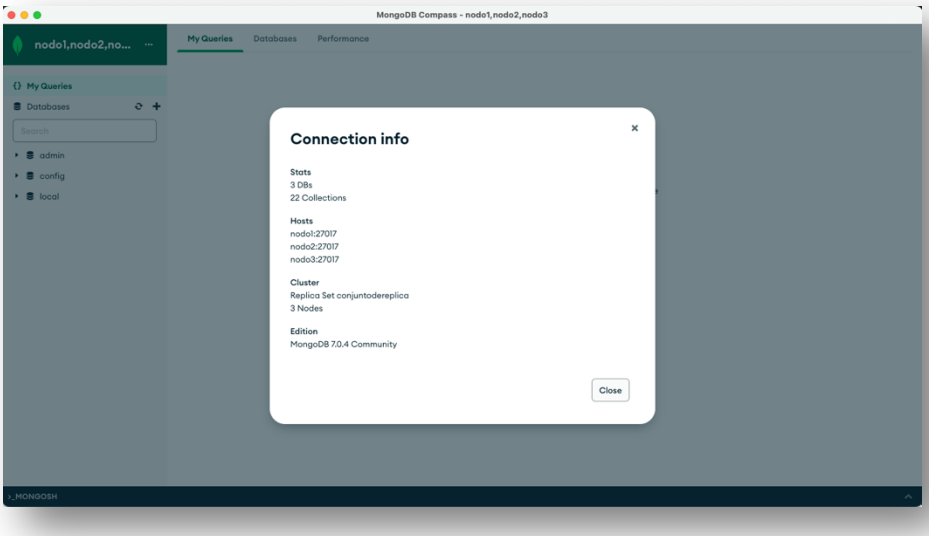
IES de Teis
Avda. de Galicia, 101
36216 – Vigo

Tfno.: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

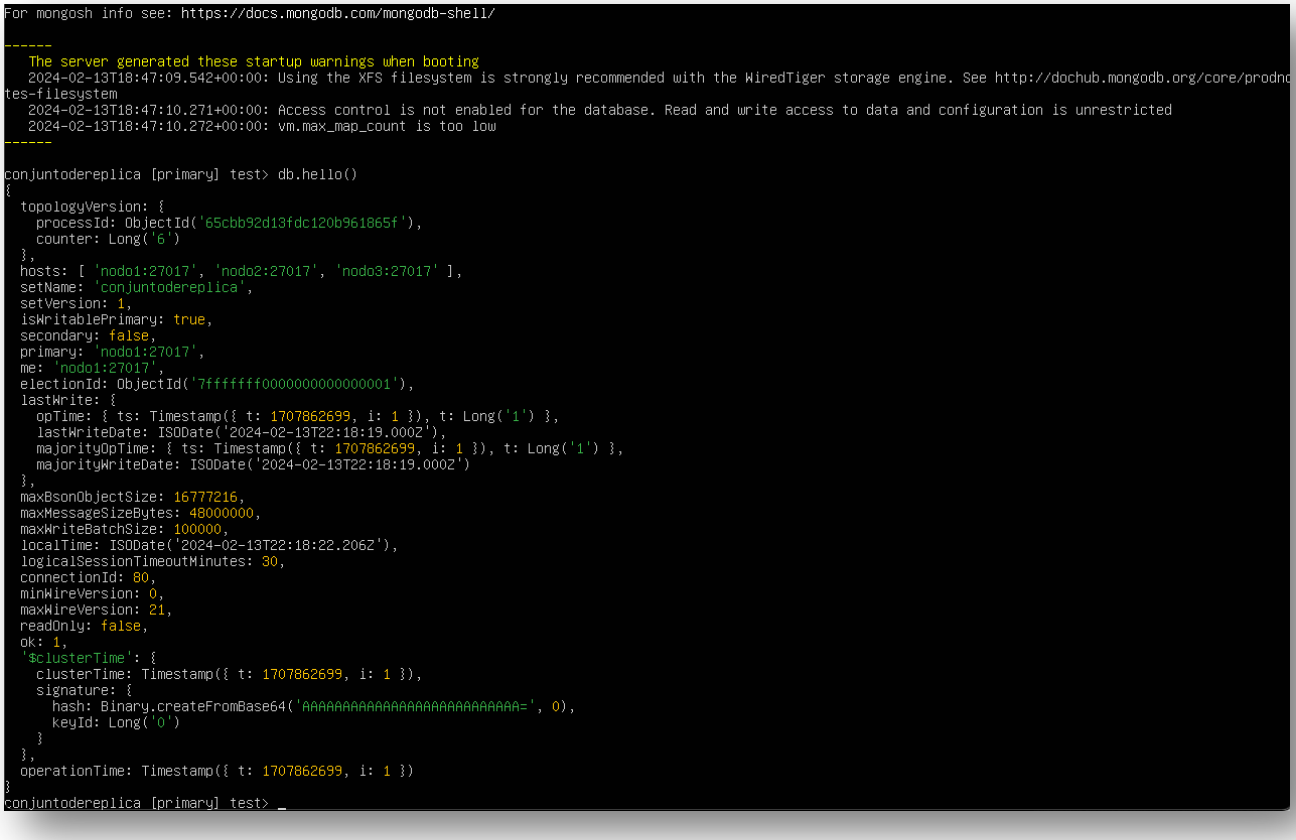

```
Mongo R3
administrador@nodo3:~$ mongosh "mongodb://nodo1:27017,nodo2:27017,nodo3:27017/?replicaSet=conjuntodereplica"
Current Mongosh Log ID: 65cbe4106d01252b9ddf7d14
Connecting to:          mongodb://nodo1:27017,nodo2:27017,nodo3:27017/?replicaSet=conjuntodereplica&appName=mongosh+2.1.0
Using MongoDB:          7.0.4
Using Mongosh:          2.1.0
mongosh 2.1.4 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-02-13T18:47:09.542+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodno
tes-filesystem
   2024-02-13T18:47:10.271+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2024-02-13T18:47:10.272+00:00: vm.max_map_count is too low
------

conjuntodereplica [primary] test> exit
administrador@nodo3:~$
administrador@nodo3:~$
administrador@nodo3:~$
administrador@nodo3:~$
administrador@nodo3:~$
administrador@nodo3:~$ mongosh "mongodb://nodo1,nodo2,nodo3"
Current Mongosh Log ID: 65cbe4199c38c7fe17ad9949
Connecting to:          mongodb://nodo1,nodo2,nodo3/?appName=mongosh+2.1.0
Using MongoDB:          7.0.4
Using Mongosh:          2.1.0
mongosh 2.1.4 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-02-13T18:47:09.542+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodno
tes-filesystem
   2024-02-13T18:47:10.271+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
   2024-02-13T18:47:10.272+00:00: vm.max_map_count is too low
------

conjuntodereplica [primary] test>
```

En la captura anterior podemos ver como desde el nodo3, que ahora mismo es un secundario, nos hemos conectado al conjunto de réplica y no a un ningún nodo en concreto. Si nos fijamos en el prompt veremos que nos ha conectado con el primario. Tal vez la única diferencia aparente es que ya no pone lo de directamente. Con la captura anterior podemos ver que hemos conectado correctamente usando una y otra cadena de conexión.

Usando un cliente como MongoDB Compass podemos usar la misma cadena de conexión para conectar al conjunto de réplica desde un cliente gráfico.



El comando "db.hello()" es útil para monitorizar el estado del conjunto de réplica , especialmente cuando programamos usando los drivers de MongoDB. Este comando devuelve mucha información como el listado de los hosts el nombre del conjunto de réplica, el nombre del primario actual o cuando se ha realizado la última escritura.



Los cambios en la configuración del conjunto de réplicas se pueden hacer en caliente sin necesidad de reiniciar nada. Para ello usaremos dos comandos, "rs.conf()" para obtener un documento con la configuración actual, y el comando "rs.reconf()" al que pasaremos un documento con la configuración que queremos.

```
rs.conf()
```

En la nueva configuración que queremos para nuestro conjunto de réplica podemos escribir de nuevo todo el documento con las opciones modificadas o bien podemos aprovechar la salida del comando "rs.conf()" para guardarlo en una variable y modificar algo mientras se mantiene el resto.

Vamos a cambiar la prioridad del nodo1 a un valor de 50 para que si alguna vez está en mantenimiento nos aseguremos que al volver vuelva a ser votado como primario. Para ello guardamos en una variable la salida del comando "rs.conf()".

```
configuracion = rs.conf()
```

Ya que trabajamos en un documento JSON podemos acceder a los campos mediante una notación con puntos.

```
configuracion.members[0].priority = 50
```

Aquí simplemente modificamos el valor prioridad del primer elemento del array de members, este elemento corresponde al nodo1.

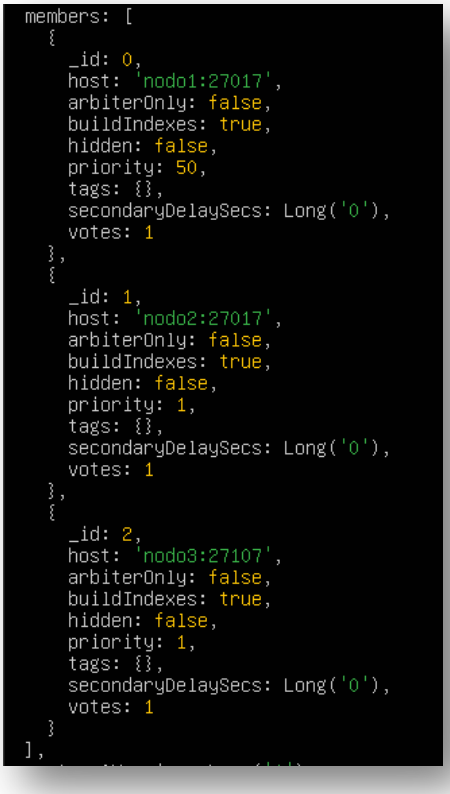
Para aplicar esta variable que almacena el documento de configuración usaremos el siguiente comando:

```
rs.reconfig(configuracion)
```

También podemos escribir nuevamente toda la configuración con el cambio que queremos hacer.

```
rs.reconfig( {
        "_id":"conjuntodereplica",
        "members":[
                {"_id":0, "host":"nodo1:27017", "priority" : 50 },
                {"_id":1, "host":"nodo2:27017" },
                {"_id":2, "host":"nodo3:27017" }
                ],
        "version": 2
        })
```

Si ahora volvemos a ejecutar el comando "rs.conf()" podremos comprobar que en la configuración actual del conjunto de réplicas ya aparece actualizado el valor de la prioridad del nodo1.



IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno.: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

Sabemos que las operaciones de escritura solo se pueden dar en el nodo primario, es una restricción. Lo que podemos modificar como preferencia es en qué momento se devuelve la confirmación de que la escritura se ha realizado permanentemente. Podemos dar por hecha la escritura en el mismo momento que se escriba en el nodo primario o bien esperar a que se replique en un número determinado de secundarios.

```
conjuntodereplica [direct: secondary] ejemplo> db.nombres.insertOne({nombre:"Laura"})
MongoServerError: not primary
conjuntodereplica [direct: secondary] ejemplo>
```

En la captura anterior podemos ver como si intentamos una escritura en un nodo secundario, este nos dará un mensaje de error en el que indica que no es un nodo primario.

En cuanto a la lectura, por defecto se realiza en el primario. Estamos ante un escenario de tolerancia a fallos puro. Podemos modificar las preferencias de lectura para que los secundarios también puedan realizar peticiones de lectura. En este caso, además de un evidente alivio para el primario, estamos ante un escenario de alta disponibilidad.

```
conjuntodereplica [direct: secondary] ejemplo> db.nombres.findOne()
MongoServerError: not primary and secondaryOk=false - consider using db.getMongo().setReadPref() or readPreference in the connection string
conjuntodereplica [direct: secondary] ejemplo>

conjuntodereplica [direct: secondary] ejemplo> db.getMongo().setReadPref("secondaryPreferred")

conjuntodereplica [direct: secondary] ejemplo>

conjuntodereplica [direct: secondary] ejemplo> db.nombres.findOne()
{ _id: ObjectId('65d24869c0cc0e3ad889e536'), nombre: 'Javi' }
conjuntodereplica [direct: secondary] ejemplo>
```

En la captura anterior podemos ver cómo, al estar conectado directamente a un secundario e intentar una operación de lectura con un "findOne", el propio secundario devuelve un mensaje de error que dice que ni es primario ni está configurado que el secundario pueda responder. En el mismo mensaje propone usar el comando "setReadPref()" para cambiar las preferencias de lectura.

```
db.getMongo().setReadPref("secondaryPreferred")
```

Con este comando cambiamos la preferencia de lectura para que se pueda leer desde los secundarios. Si revisamos la captura anterior podemos ver la secuencia completa en la que se ha intentado una lectura y ha fallado, hemos modificado la preferencia de lectura y después hemos vuelto a intentar la escritura. En este intento comprobamos que ya funciona.

Es importante tener claro que las lecturas no necesariamente mostrarán datos reales en el sentido de datos actualizados.